

**SYSTEM, METHOD AND ARTICLE OF MANUFACTURE FOR A
BUSINESS LAYER COMPONENT IN A MULTIMEDIA
SYNCHRONIZATION FRAMEWORK**

5

FIELD OF THE INVENTION

The present invention relates to network synchronization and more particularly to software for synchronizing the playback of a multimedia event on a plurality of client apparatuses.

10

BACKGROUND OF THE INVENTION

15 Systems such as the Internet typically are point-to-point (or unicast) systems in which a message is converted into a series of addressed packets which are routed from a source node through a plurality of routers to a destination node. In most communication protocols the packet includes a header which contains the addresses of the source and the destination nodes as well as a sequence number which specifies the packet's order in the message.

20

In general, these systems do not have the capability of broadcasting a message from a source node to all the other nodes in the network because such a capability is rarely of much use and could easily overload the network. However, there are situations where it is desirable for one node to communicate with some subset of all the nodes. For example, multi-party conferencing capability analogous to that found in the public telephone system and broadcasting to a limited number of nodes are of considerable interest to users of packet-switched networks. To satisfy such demands, packets destined for several recipients have been encapsulated in a unicast packet and forwarded from a source to a point in a network where the packets have been replicated and forwarded on to all desired recipients. This technique is known as IP Multicasting and the network over which such packets are routed is referred to

25

30

09489601.012000

as the Multicast Backbone or MBONE. More recently, routers have become available which can route the multicast addresses (class D addresses) provided for in communication protocols such as TCP/IP and UDP/IP. A multicast address is essentially an address for a group of host computers who have indicated their desire to participate in that group. Thus, a multicast packet can be routed from a source node through a plurality of multicast routers (or mrouters) to one or more devices receiving the multicast packets. From there the packet is distributed to all the host computers that are members of the multicast group.

10 These techniques have been used to provide on the Internet audio and video conferencing as well as radio-like broadcasting to groups of interested parties. See, for example, K. Savetz et al. MBONE Multicasting Tomorrow's Internet (IDG Books WorldWide Inc., 1996).

15 Further details concerning technical aspects of multicasting may be found in the Internet documents Request for Comments (RFC) 1112 and 1458 which are reproduced at Appendices A and B of the Savetz book and in D. P. Brutaman et al., "MBONE provides Audio and Video Across the Internet," IEEE Computer, Vol. 27, No. 4, pp. 30-36 (April 1994), all of which are incorporated herein by reference.

20 Multimedia computer systems have become increasingly popular over the last several years due to their versatility and their interactive presentation style. A multimedia computer system can be defined as a computer system having a combination of video and audio outputs for presentation of audio-visual displays. A
25 modem multimedia computer system typically includes one or more storage devices such as an optical drive, a CD-ROM, a hard drive, a videodisc, or an audiodisc, and audio and video data are typically stored on one or more of these mass storage devices. In some file formats the audio and video are interleaved together in a single file, while in other formats the audio and video data are stored in different files,
30 many times on different storage media. Audio and video data for a multimedia display may also be stored in separate computer systems that are networked together. In this instance, the computer system presenting the multimedia display would

receive a portion of the necessary data from the other computer system via the network cabling.

Graphic images used in Windows multimedia applications can be created in either of two ways, these being bit-mapped images and vector-based images. Bit-mapped images comprise a plurality of picture elements (pixels) and are created by assigning a color to each pixel inside the image boundary. Most bit-mapped color images require one byte per pixel for storage, so large bit-mapped images create correspondingly large files. For example, a full-screen, 256-color image in 640-by-480-pixel VGA mode requires 307,200 bytes of storage, if the data is not compressed. Vector-based images are created by defining the end points, thickness, color, pattern and curvature of lines and solid objects comprised within the image. Thus, a vector-based image includes a definition which consists of a numerical representation of the coordinates of the object, referenced to a corner of the image.

Bit-mapped images are the most prevalent type of image storage format, and the most common bit-mapped-image file formats are as follows. A file format referred to as BMP is used for Windows bit-map files in 1-, 2-, 4-, 8-, and 24-bit color depths. BMP files contain a bit-map header that defines the size of the image, the number of color planes, the type of compression used (if any), and the palette used. The Windows DIB (device-independent bit-map) format is a variant of the BMP format that includes a color table defining the RGB (red green blue) values of the colors used. Other types of bit-map formats include the TIF (tagged image format file), the PCX (Zsoft Personal Computer Paintbrush Bitmap) file format, the GIF (graphics interchange file) format, and the TGA (Texas Instruments Graphic Architecture) file format.

The standard Windows format for bit-mapped images is a 256-color device-independent bit map (DIB) with a BMP (the Windows bit-mapped file format) or sometimes a DIB extension. The standard Windows format for vector-based images is referred to as WMF (Windows meta file).

Full-motion video implies that video images shown on the computer's screen simulate those of a television set with identical (30 frames-per-second) frame rates, and that these images are accompanied by high-quality stereo sound. A large amount of storage is required for high-resolution color images, not to mention a full-motion video sequence. For example, a single frame of NTSC video at 640-by-400-pixel resolution with 16-bit color requires 512K of data per frame. At 30 flames per second, over 15 Megabytes of data storage are required for each second of full motion video. Due to the large amount of storage required for full motion video, various types of video compression algorithms are used to reduce the amount of necessary storage. Video compression can be performed either in real-time, i.e., on the fly during video capture, or on the stored video file after the video data has been captured and stored on the media. In addition, different video compression methods exist for still graphic images and for full-motion video.

Examples of video data compression for still graphic images are RLE (run-length encoding) and JPEG (Joint Photographic Experts Group) compression. RLE is the standard compression method for Windows BMP and DIB files. The RLE compression method operates by testing for duplicated pixels in a single line of the bit map and stores the number of consecutive duplicate pixels rather than the data for the pixel itself. JPEG compression is a group of related standards that provide either lossless (no image quality degradation) or lossy (imperceptible to severe degradation) compression types. Although JPEG compression was designed for the compression of still images rather than video, several manufacturers supply JPEG compression adapter cards for motion video applications.

25

In contrast to compression algorithms for still images, most video compression algorithms are designed to compress full motion video. Video compression algorithms for motion video generally use a concept referred to as interframe compression, which involves storing only the differences between successive frames in the data file. Interframe compression begins by digitizing the entire image of a key frame. Successive frames are compared with the key frame, and only the differences between the digitized data from the key frame and from the successive

frames are stored. Periodically, such as when new scenes are displayed, new key frames are digitized and stored, and subsequent comparisons begin from this new reference point. It is noted that interframe compression ratios are content-dependent, i.e., if the video clip being compressed includes many abrupt scene transitions from one image to another, the compression is less efficient. Examples of video compression which use an interframe compression technique are MPEG, DVI and Indeo, among others.

MPEG (Moving Pictures Experts Group) compression is a set of methods for compression and decompression of full motion video images that uses the interframe compression technique described above. The MPEG standard requires that sound be recorded simultaneously with the video data, and the video and audio data are interleaved in a single file to attempt to maintain the video and audio synchronized during playback. The audio data is typically compressed as well, and the MPEG standard specifies an audio compression method referred to as ADPCM (Adaptive Differential Pulse Code Modulation) for audio data.

A standard referred to as Digital Video Interactive (DVI) format developed by Intel Corporation is a compression and storage format for full-motion video and high-fidelity audio data. The DVI standard uses interframe compression techniques similar to that of the MPEG standard and uses ADPCM compression for audio data. The compression method used in DVI is referred to as RTV 2.0 (real time video), and this compression method is incorporated into Intel's AVK (audio/video kernel) software for its DVI product line. IBM has adopted DVI as the standard for displaying video for its Ultimedia product line. The DVI file format is based on the Intel i750 chipset and is supported through the Media Control Interface (MCI) for Windows. Microsoft and Intel jointly announced the creation of the DV MCI (digital video media control interface) command set for Windows 3.1 in 1992.

The Microsoft Audio Video Interleaved (AVI) format is a special compressed file structure format designed to enable video images and synchronized sound stored on CD-ROMs to be played on PCs with standard VGA displays and audio adapter

cards. The AVI compression method uses an interframe method, i.e., the differences between successive frames are stored in a manner similar to the compression methods used in DVI and MPEG. The AVI format uses symmetrical software compression-decompression techniques, i.e., both compression and decompression
5 are performed in real time. Thus AVI files can be created by recording video images and sound in AVI format from a VCR or television broadcast in real time, if enough free hard disk space is available.

10 Despite these compression algorithms, it is very difficult to simultaneously multicast multimedia material due to bandwidth restraints. This problem is unavoidable with present technology since such large amounts of data must be transferred over networks such as the Internet from a single host server to numerous client computers.

SUMMARY OF THE INVENTION

A system, method and article of manufacture are provided for identifying a plurality of events which are played back simultaneously on a plurality of networked client apparatuses. First, a plurality of events are stored in memory on a plurality of client apparatuses. The events each have a unique identifier associated therewith and which are stored in the memory. In operation, the client apparatuses are adapted to be coupled to a host computer via a network. The identifier of the event which is stored in the memory of the client apparatuses is then retrieved utilizing the network. Such identifier is subsequently compared with an identifier of a scheduled event. If the comparison renders a match, the playback of the event is begun on each of the client apparatuses.

09489601.012000

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be better understood when consideration is given to the following detailed description thereof. Such description makes reference to the annexed
5 drawings wherein:

Figure 1 is a schematic diagram of a hardware implementation of one embodiment of the present invention;

10 Figure 2 illustrates a flowchart delineating a method for synchronizing an event on a plurality of client apparatuses in accordance with one embodiment of the present invention;

Figure 3 illustrates a flowchart delineating a method for storing synchronization
15 information for subsequent playback of an event in accordance with one embodiment of the present invention;

Figure 4 illustrates a flowchart setting forth a method for providing overlays during a
20 synchronized event on a plurality of client apparatuses in accordance with one embodiment of the present invention;

Figure 5 illustrates a flow diagram for delayed synchronization of an event on a
25 plurality of client apparatuses in accordance with one embodiment of the present invention;

Figure 6 illustrates a flow diagram for providing information on a synchronized
event on a plurality of client apparatuses in accordance with one embodiment of the
present invention;

30 Figure 7 illustrates a method for creating a synchronizer object in order to playback an event simultaneously on a plurality of client apparatuses in accordance with one embodiment of the present invention;

09439501.012000

Figure 8 illustrates a flowchart for affording a scheduler object adapted to facilitate the playback of an event simultaneously on a plurality of networked client apparatuses in accordance with one embodiment of the present invention;

5

Figure 9 is a flowchart delineating a method for identifying a plurality of events which are played back simultaneously on a plurality of networked client apparatuses in accordance with one embodiment of the present invention;

10 Figure 10 shows a flowchart delineating a technique for interfacing a plurality of different types of playback devices of client apparatuses which are networked to simultaneously playback an event in accordance with one embodiment of the present invention;

15 Figure 11 illustrates the manner in which a layer factory is created in accordance with one embodiment of the present invention;

Figure 12 illustrates the manner in which user requests are processed in accordance with one embodiment of the present invention;

20

Figures 13-16 illustrate various class/component diagrams in accordance with one embodiment of the present invention;

Figure 17 illustrates a logical sequence diagram in accordance with one embodiment of the present invention;

25

Figure 18 illustrates a logical sequence diagram that shows server side collaboration in accordance with one embodiment of the present invention; and

30 Figure 19 illustrates a logical sequence diagram showing client side collaboration in accordance with one embodiment of the present invention.

09489601-012000

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figures 1-19 illustrate a system for synchronizing an event on a plurality of client apparatuses. Prior to use, an event is stored in memory on at least one of the client apparatuses. Such client apparatuses are adapted to be connected to a network along with a host computer(s). In operation, information is transmitted from the host computer to the at least one client apparatus utilizing the network. This information allows for the simultaneous and synchronous playback of the event on each of the client apparatuses.

In various embodiments, the client apparatuses may take the form of computers, televisions, stereos, home appliances, or any other types of devices. In one embodiment, the client apparatuses and the host computer each include a computer such as an IBM compatible computer, Apple Macintosh computer or UNIX based workstation.

A representative hardware environment is depicted in Figure 1, which illustrates a typical hardware configuration of a workstation in accordance with a preferred embodiment having a central processing unit 110, such as a microprocessor, and a number of other units interconnected via a system bus 112. The workstation shown in Figure 1 includes a Random Access Memory (RAM) 114, Read Only Memory (ROM) 116, an I/O adapter 118 for connecting peripheral devices such as disk storage units 120 (i.e. DVD playback device) to the bus 112, a user interface adapter 122 for connecting a keyboard 124, a mouse 126, a speaker 128, a microphone 132, and/or other user interface devices such as a touch screen (not shown) to the bus 112, communication adapter 134 for connecting the workstation to a communication network (e.g., a data processing network) and a display adapter 136 for connecting the bus 112 to a display device 138. The workstation typically has resident thereon an operating system such as the Microsoft Windows NT or Windows/95 Operating System (OS), the IBM OS/2 operating system, the MAC OS, or UNIX operating system. Those skilled in the art will appreciate that the present invention may also be implemented on platforms and operating systems other than those mentioned.

A preferred embodiment is written using JAVA, C, and the C++ language and utilizes object oriented programming methodology. Object oriented programming (OOP) has become increasingly used to develop complex applications. As OOP
5 moves toward the mainstream of software design and development, various software solutions require adaptation to make use of the benefits of OOP. A need exists for these principles of OOP to be applied to a messaging interface of an electronic messaging system such that a set of OOP classes and objects for the messaging interface can be provided.

10

OOP is a process of developing computer software using objects, including the steps of analyzing the problem, designing the system, and constructing the program. An object is a software package that contains both data and a collection of related structures and procedures. Since it contains both data and a collection of structures
15 and procedures, it can be visualized as a self-sufficient component that does not require other additional structures, procedures or data to perform its specific task. OOP, therefore, views a computer program as a collection of largely autonomous components, called objects, each of which is responsible for a specific task. This concept of packaging data, structures, and procedures together in one component or
20 module is called encapsulation.

In general, OOP components are reusable software modules which present an interface that conforms to an object model and which are accessed at run-time through a component integration architecture. A component integration architecture
25 is a set of architecture mechanisms which allow software modules in different process spaces to utilize each others capabilities or functions. This is generally done by assuming a common component object model on which to build the architecture. It is worthwhile to differentiate between an object and a class of objects at this point. An object is a single instance of the class of objects, which is often just called a
30 class. A class of objects can be viewed as a blueprint, from which many objects can be formed.

OOP allows the programmer to create an object that is a part of another object. For example, the object representing a piston engine is said to have a composition-relationship with the object representing a piston. In reality, a piston engine comprises a piston, valves and many other components; the fact that a piston is an
5 element of a piston engine can be logically and semantically represented in OOP by two objects.

OOP also allows creation of an object that "depends from" another object. If there are two objects, one representing a piston engine and the other representing a piston
10 engine wherein the piston is made of ceramic, then the relationship between the two objects is not that of composition. A ceramic piston engine does not make up a piston engine. Rather it is merely one kind of piston engine that has one more limitation than the piston engine; its piston is made of ceramic. In this case, the object representing the ceramic piston engine is called a derived object, and it
15 inherits all of the aspects of the object representing the piston engine and adds further limitation or detail to it. The object representing the ceramic piston engine "depends from" the object representing the piston engine. The relationship between these objects is called inheritance.

20 When the object or class representing the ceramic piston engine inherits all of the aspects of the objects representing the piston engine, it inherits the thermal characteristics of a standard piston defined in the piston engine class. However, the ceramic piston engine object overrides these ceramic specific thermal characteristics, which are typically different from those associated with a metal piston. It skips over
25 the original and uses new functions related to ceramic pistons. Different kinds of piston engines have different characteristics, but may have the same underlying functions associated with it (e.g., how many pistons in the engine, ignition sequences, lubrication, etc.). To access each of these functions in any piston engine object, a programmer would call the same functions with the same names, but each
30 type of piston engine may have different/overriding implementations of functions behind the same name. This ability to hide different implementations of a function

behind the same name is called polymorphism and it greatly simplifies communication among objects.

With the concepts of composition-relationship, encapsulation, inheritance and polymorphism, an object can represent just about anything in the real world. In fact, one's logical perception of the reality is the only limit on determining the kinds of things that can become objects in object-oriented software. Some typical categories are as follows:

- Objects can represent physical objects, such as automobiles in a traffic-flow simulation, electrical components in a circuit-design program, countries in an economics model, or aircraft in an air-traffic-control system.
- Objects can represent elements of the computer-user environment such as windows, menus or graphics objects.
- An object can represent an inventory, such as a personnel file or a table of the latitudes and longitudes of cities.
- An object can represent user-defined data types such as time, angles, and complex numbers, or points on the plane.

With this enormous capability of an object to represent just about any logically separable matters, OOP allows the software developer to design and implement a computer program that is a model of some aspects of reality, whether that reality is a physical entity, a process, a system, or a composition of matter. Since the object can represent anything, the software developer can create an object which can be used as a component in a larger software project in the future.

If 90% of a new OOP software program consists of proven, existing components made from preexisting reusable objects, then only the remaining 10% of the new software project has to be written and tested from scratch. Since 90% already came from an inventory of extensively tested reusable objects, the potential domain from which an error could originate is 10% of the program. As a result, OOP enables software developers to build objects out of other, previously built objects.

This process closely resembles complex machinery being built out of assemblies and sub-assemblies. OOP technology, therefore, makes software engineering more like hardware engineering in that software is built from existing components, which are
5 available to the developer as objects. All this adds up to an improved quality of the software as well as an increased speed of its development.

Programming languages are beginning to fully support the OOP principles, such as encapsulation, inheritance, polymorphism, and composition-relationship. With the
10 advent of the C++ language, many commercial software developers have embraced OOP. C++ is an OOP language that offers a fast, machine-executable code. Furthermore, C++ is suitable for both commercial-application and systems-programming projects. For now, C++ appears to be the most popular choice among many OOP programmers, but there is a host of other OOP languages, such as
15 Smalltalk, Common Lisp Object System (CLOS), and Eiffel. Additionally, OOP capabilities are being added to more traditional popular computer programming languages such as Pascal.

The benefits of object classes can be summarized, as follows:

- 20 • Objects and their corresponding classes break down complex programming problems into many smaller, simpler problems.
- Encapsulation enforces data abstraction through the organization of data into small, independent objects that can communicate with each other. Encapsulation protects the data in an object from accidental damage, but
25 allows other objects to interact with that data by calling the object's member functions and structures.
- Subclassing and inheritance make it possible to extend and modify objects through deriving new kinds of objects from the standard classes available in the system. Thus, new capabilities are created without having to start from
30 scratch.

programmers from the chores involved in displaying menus, windows, dialog boxes, and other standard user interface elements for personal computers.

Frameworks also represent a change in the way programmers think about the
5 interaction between the code they write and code written by others. In the early days
of procedural programming, the programmer called libraries provided by the
operating system to perform certain tasks, but basically the program executed down
the page from start to finish, and the programmer was solely responsible for the flow
of control. This was appropriate for printing out paychecks, calculating a
10 mathematical table, or solving other problems with a program that executed in just
one way.

The development of graphical user interfaces began to turn this procedural
programming arrangement inside out. These interfaces allow the user, rather than
15 program logic, to drive the program and decide when certain actions should be
performed. Today, most personal computer software accomplishes this by means of
an event loop which monitors the mouse, keyboard, and other sources of external
events and calls the appropriate parts of the programmer's code according to actions
that the user performs. The programmer no longer determines the order in which
20 events occur. Instead, a program is divided into separate pieces that are called at
unpredictable times and in an unpredictable order. By relinquishing control in this
way to users, the developer creates a program that is much easier to use.

Nevertheless, individual pieces of the program written by the developer still call
libraries provided by the operating system to accomplish certain tasks, and the
25 programmer must still determine the flow of control within each piece after it's
called by the event loop. Application code still "sits on top of" the system.

Even event loop programs require programmers to write a lot of code that should not
need to be written separately for every application. The concept of an application
30 framework carries the event loop concept further. Instead of dealing with all the
nuts and bolts of constructing basic menus, windows, and dialog boxes and then
making these things all work together, programmers using application frameworks

be combined, including rules for what a programmer is supposed to provide versus what the framework provides.

- Call versus override. With a class library, the code the programmer instantiates objects and calls their member functions. It's possible to instantiate and call objects in the same way with a framework (i.e., to treat the framework as a class library), but to take full advantage of a framework's reusable design, a programmer typically writes code that overrides and is called by the framework. The framework manages the flow of control among its objects. Writing a program involves dividing responsibilities among the various pieces of software that are called by the framework rather than specifying how the different pieces should work together.
- Implementation versus design. With class libraries, programmers reuse only implementations, whereas with frameworks, they reuse design. A framework embodies the way a family of related programs or pieces of software work. It represents a generic design solution that can be adapted to a variety of specific problems in a given domain. For example, a single framework can embody the way a user interface works, even though two different user interfaces created with the same framework might solve quite different interface problems.

Thus, through the development of frameworks for solutions to various problems and programming tasks, significant reductions in the design and development effort for software can be achieved. A preferred embodiment of the invention utilizes HyperText Markup Language (HTML) to implement documents on the Internet together with a general-purpose secure communication protocol for a transport medium between the client and the Newco. HTTP or other protocols could be readily substituted for HTML without undue experimentation. Information on these products is available in T. Berners-Lee, D. Connolly, "RFC 1866: Hypertext Markup Language - 2.0" (Nov. 1995); and R. Fielding, H. Frystyk, T. Berners-Lee, J. Gettys and J.C. Mogul, "Hypertext Transfer Protocol -- HTTP/1.1: HTTP Working Group Internet Draft" (May 2, 1996). HTML is a simple data format used to create hypertext documents that are portable from one platform to another. HTML

documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of domains. HTML has been in use by the World-Wide Web global information initiative since 1990. HTML is an application of ISO Standard 8879; 1986 Information Processing Text and Office Systems; Standard Generalized Markup Language (SGML).

To date, Web development tools have been limited in their ability to create dynamic Web applications which span from client to server and interoperate with existing computing resources. Until recently, HTML has been the dominant technology used in development of Web-based solutions. However, HTML has proven to be inadequate in the following areas:

- Poor performance;
- Restricted user interface capabilities;
- Can only produce static Web pages;
- Lack of interoperability with existing applications and data; and
- Inability to scale.

Sun Microsystem's Java language solves many of the client-side problems by:

- Improving performance on the client side;
- Enabling the creation of dynamic, real-time Web applications; and
- Providing the ability to create a wide variety of user interface components.

With Java, developers can create robust User Interface (UI) components. Custom "widgets" (e.g., real-time stock tickers, animated icons, etc.) can be created, and client-side performance is improved. Unlike HTML, Java supports the notion of client-side validation, offloading appropriate processing onto the client for improved performance. Dynamic, real-time Web pages can be created. Using the above-mentioned custom UI components, dynamic Web pages can also be created.

Sun's Java language has emerged as an industry-recognized language for "programming the Internet." Sun defines Java as: "a simple, object-oriented,

distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, dynamic, buzzword-compliant, general-purpose programming language. Java supports programming for the Internet in the form of platform-independent Java applets." Java applets are small, specialized applications that comply with Sun's Java Application Programming Interface (API) allowing developers to add "interactive content" to Web documents (e.g., simple animations, page adornments, basic games, etc.). Applets execute within a Java-compatible browser (e.g., Netscape Navigator) by copying code from the server to client. From a language standpoint, Java's core feature set is based on C++. Sun's Java literature states that Java is basically, "C++ with extensions from Objective C for more dynamic method resolution."

Another technology that provides similar function to JAVA is provided by Microsoft and ActiveX Technologies, to give developers and Web designers wherewithal to build dynamic content for the Internet and personal computers. ActiveX includes tools for developing animation, 3-D virtual reality, video and other multimedia content. The tools use Internet standards, work on multiple platforms, and are being supported by over 100 companies. The group's building blocks are called ActiveX Controls, small, fast components that enable developers to embed parts of software in hypertext markup language (HTML) pages. ActiveX Controls work with a variety of programming languages including Microsoft Visual C++, Borland Delphi, Microsoft Visual Basic programming system and, in the future, Microsoft's development tool for Java, code named "Jakarta." ActiveX Technologies also includes ActiveX Server Framework, allowing developers to create server applications. One of ordinary skill in the art readily recognizes that ActiveX could be substituted for JAVA without undue experimentation to practice the invention.

Synchronization Overview

Figure 2 illustrates a flowchart delineating a method for synchronizing an event on a plurality of client apparatuses. First, in operation 200, an event is stored in memory on at least one of the client apparatuses. In various embodiments, the memory may

take the form of an electromagnetic medium, or any type of optical storage device, i.e. CD-audio. In a primary aspect of the present invention, the memory includes a digital video disc (DVD) (audio or video). Further, for reasons that will soon become apparent, the information includes chapter information associated with the DVD. In such embodiment where the memory is portable, the user may be required to purchase the memory, i.e. DVD, in order to participate in a synchronized event, thus increasing the sale of DVD's.

It should be noted that the event need not be necessarily stored in memory on all of the client apparatuses, but rather stored on one or some of the client apparatuses and streamed to the remaining client apparatuses at variant rates. This may be feasibly accomplished if the client apparatus(es) containing the stored event has a high-bandwidth connection with the remaining client apparatuses. For example, the client apparatus(es) containing the stored event may include a server that has a connection to a plurality of televisions via a cable network, i.e. WEBTV. Similar functionality may be achieved via a broadcast medium. The present invention is thus flexible by having an ability to host user events and corporate events.

In one embodiment, the event includes a video and audio presentation such as movie, a concert, and/or a theatrical event. It should be noted, however, that the event may included any recording capable of being played back for entertainment, education, informative or other similar purposes.

In use, the client apparatuses and a host computer are adapted to be connected to a network. Such network may include a wide, local or any other type of communication network. For example, a wide area network such as the Internet may be employed which operates using TCP/IP or IPX protocols.

In operation 202, information is transmitted from the host computer to the appropriate client apparatuses utilizing the network. This information allows for the simultaneous and synchronous playback of the event on each of the client apparatuses. In one embodiment, the information may also include a start time when

the playback of the event is to begin on each of the client apparatuses. Further, an ending time may be included when the playback of the event is to end on each of the client apparatuses. Still yet, “play” command information may be sent to the client apparatuses at the start time. As an option, input may be received from the user, and used to alter the playback of the event. The host server, or synchronization server, can also control various streams of a variant rate and different hardware associated with those streams.

The present invention thus has the ability to synchronize video playback for one or multiple (thousands) users from one or multiple physical locations, and to synchronize with external video, audio and/or data streams.

Users of the present invention are at multiple physical locations and host servers may also be at different locations. The present invention is thus a scalable system which is capable of servicing an unlimited number of users. Since the content is local to the user machine, no high network bandwidth is required.

History Download Capabilities

Figure 3 illustrates a flowchart delineating a method for storing synchronization information for subsequent playback of an event. Initially, in operation 300, an event is stored in memory on at least one of the client apparatuses, as set forth earlier. These client apparatuses are adapted to be connected to a network along with a host computer during use.

In operation 302, information is stored on the host computer(s) for allowing the simultaneous playback of the event on each of the client apparatuses. In one embodiment, the information may include a history and data associated with the synchronous playback. In particular, the history may include any overlaid material(as will be described hereinafter in greater detail), any specific commands affecting the playback of the information, or any other type of general information, i.e. start time, end time, etc.

30

In operation 304, the information may be downloaded utilizing the network at any time after the synchronous playback of the event. Such downloaded information may then be used for playback after the simultaneous playback of the event. As
5 such, the present invention has the ability to allow users to download a history and data associated with a particular synchronization event and play it later.

Overlay Synchronization

10 Figure 4 illustrates a flowchart setting forth a method for providing overlays during a synchronized event on a plurality of client apparatuses or any other source. First, in operation 400, a plurality of client apparatuses are connected via a network. In operation 402, an event may be simultaneously played back on the client apparatuses utilizing the network, as set forth earlier.

15 During the playback of the event, visual and/or audio material may also be overlaid on the event based on input received from at least one of the client apparatuses. See operation 404. This may be accomplished by transmitting the overlay material from one of the client apparatuses to the host computer or any other server, and
20 multicasting the same to the remaining client apparatuses.

As an option, the overlay material may include annotations on a display of the client apparatus. For example, the overlay material may include sketches which are inputted by way of a stylus-based input screen or a keyboard or the like, along with a
25 voiceover inputted by way of a microphone or voice synthesizer. Such capability may also be quite valuable in an educational environment.

In one embodiment, the overlay material may also be displayed on each of the client apparatuses utilizing the network. This allows each of the users to experience the
30 overlay in real-time during the simultaneous playback of the event. As an option, the user inputting the overlay material may select which users may experience the

overlay material. The client apparatus that provided the overlay material may also be identified to the users experiencing the overlay material.

It should be noted that various bi-directional communication may be enabled for
5 allowing data to travel to and from the server. For instance, the playback of the event on the client apparatuses may be altered in any feasible way based on input from a user.

Late Synchronization

10 Figure 5 illustrates a flow diagram for delayed synchronization of an event on a plurality of client apparatuses. First, in operation 500, a plurality of client apparatuses are connected via a network and an event is stored in memory on the client apparatuses. The event is then simultaneously played back on the client apparatuses utilizing the network, as set forth earlier. Note operation 502.

15 During the simultaneous playback, a request may be received from one of the client apparatuses for that particular to be included in the synchronized event, as set forth in operation 504. This request may be received after the synchronized event has already begun while it is still playing. Further, the request may be submitted via a
20 site on a network, i.e. website.

In response to the request, information is transmitted in operation 506 to the requesting client apparatus utilizing the network. This information is adapted for identifying a location in the memory where the event is currently being played back.
25 This allows the simultaneous playback of the event on the requesting client apparatus.

The end users are thus able to come in at a later time and to be synchronized with the event. Targeted synchronization and various filters criteria can be applied to target
30 different audiences. Also language and cultural differences can be taken into account. Still yet, the present invention may be adapted to address users on different hardware platforms (MAC, PC, set-top boxes). This may be accomplished by

and retailer) are able to add tracking information to BCA database. Due to complexity and dependencies on existing business systems, the retail tracking concept will be rolled out in phases: replicator first most likely with key retail accounts. The distributors will be brought in. Retailers will then begin to embrace the ability to track based on local outlet/store.

By the foregoing design, easy deployment is thus afforded and minimal hardware is required to allow the synchronization of content without significant capital investments and with a very efficient control mechanism. The content delivery does not rely on high network bandwidth and is independent from the synchronization.

Internet Server Application Program Interface (ISAPI) extensions will be used on the server. ISAPI extensions provide a mechanism to maintain a temporary or permanent connection with the users. These connections allow the Synchronization Server to process request and to send the appropriate DVD commands. The permanent connections are known as "Keep Alive" connections. ISAPI extension can also be used as an HTTP interface to a more traditional server, with all data returned as text.

On the client side the approach is to use, but not limited to Java 1.1 applets, to initiate event start-up for the Synchronization server. The advantage of using Java 1.1 applets is to achieve platform independence for existing and future Java-enabled devices. JavaScript will be used to provide user interface navigation by "wrapping" the applet.

An ISAPI (Internet Server Application Program Interface) is a set of Windows program calls that let one write a Web server application that will run faster than a Common Gateway Interface (CGI) application. A disadvantage of a CGI application (or "executable file," as it is sometimes called) is that each time it is run, it runs as a separate process with its own address space, resulting in extra instructions that have to be performed, especially if many instances of it are running on behalf of users.

Using ISAPI, you create a Dynamic Link Library (DLL) application file that can run as part of the Hypertext Transport Protocol (HTTP) application's process and address space. The DLL files are loaded into the computer when HTTP is started and remain there as long as they are needed; they don't have to be located and read into storage as frequently as a CGI application.

Existing CGI applications can be converted into ISAPI application DLLs without having to rewrite their logic. However, they do need to be written to be thread-safe so that a single instance of the DLL can serve multiple users.

A special kind of ISAPI DLL is called an ISAPI filter, which can be designated to receive control for every HTTP request. One can create an ISAPI filter for encryption or decryption, for logging, for request screening, or for other purposes.

One can write ISAPI server extension DLLs (ISAs) that can be loaded and called by the HTTP server. Users can fill out forms and click a submit button to send data to a Web server and invoke an ISA, which can process the information to provide custom content or store it in a database. Web server extensions can use information in a database to build Web pages dynamically, and then send them to the client computers to be displayed. An application can add other custom functionality and provide data to the client using HTTP and HTML.

One can write an ISAPI filter. The filter is also a DLL that runs on an ISAPI-enabled HTTP server. The filter registers for notification of events such as logging on or URL mapping. When the selected events occur, the filter is called, and one can monitor and change the data (on its way from the server to the client or vice versa). ISAPI filters can be used to provide custom encryption or compression schemes, or additional authentication methods.

Both server extensions and filters run in the process space of the Web server, providing an efficient way to extend the server's capabilities.

Overall Component Design

0000210" T0968460

The various functional components of the software associated with the present invention will now be set forth. Such components include a Java/JavaScript Component, Synchronizer Component, LayerImpl Component, Business Layer
5 Component, Configuration Manager Component, and DBConnect Component.

Java/JavaScript Component

Figure 6 illustrates a flow diagram for providing information on a synchronized
10 event on a plurality of client apparatuses in accordance with one embodiment of the present invention. First, in operation 600, a plurality of client apparatuses are connected via a network, as set forth earlier. Next, an application program is embedded on a site on the network in operation 602. Such application program may take the form of a JAVA applet, and the site may include a website on the Internet.

15 In use, information is requested from a server on the network utilizing the application program. See operation 604. Such information relates to an event to be played back simultaneously on the client apparatuses and may include general information such as a start and stop time of the event, or more specific information
20 about the event itself.

In response to such request, a script is received for displaying the information. Note operation 606. The script may take any form such as Perl, REXX (on IBM mainframes), and Tcl/Tk, and preferably includes a JAVAScript.

25 In one embodiment of the present invention, the JAVA applet may be further adapted to send a request to retrieve command information from the server for use with a playback device of one of the client apparatuses. The commands may be adapted to playback the event on the playback device simultaneous with the
30 playback of the event on the remaining client apparatuses. Further, the commands may include a start time when the playback of the event is to begin on each of the client apparatuses.

In summary, when the Synchronizer component receives a "*BroadcastVideoEvent*" from the applet, it then places the request in the thread queue for processing. To process a request, the thread creates a "call back" object, if one does not exist for this event. The thread then adds the request to the "call back" object queue. This "call back" object will be invoked when it is time to play the DVD. The Synchronizer component creates a Call Back COM object, *LayerSink*. The Synchronizer component is also responsible for creating the *LayerFactory* interface which will be set forth hereinafter in greater detail.

LayerImpl Component

Figure 8 illustrates a flowchart for affording a scheduler object adapted to facilitate the playback of an event simultaneously on a plurality of networked client apparatuses. The present method ensures that critical information is tracked during the synchronization of the event. Such critical information not only ensures proper synchronization, but also enables various peripheral features.

First, in operation 800, various values are determined including a current time, a start time when an event is to start, and a stop time when the event is to end. Thereafter, a length of the event is calculated based on the start time and the stop time in operation 802. As an option, the current time is determined by querying a clock of one of the client apparatuses.

If any portion of the length of the event takes place during a predetermined threshold period, a command is stored in memory in operation 804. The command may be adapted to automatically begin playing back the event at the start time. In one embodiment, the threshold period includes the time the users can be queued before the event. As an option, chapter information may be stored in the memory if any portion of the length of the event takes place during the predetermined threshold period. This allows the command to automatically begin playing back the event at a predetermined chapter.

In operation 806, a loop is created at the start time during which a lapsed time of the event is tracked. This information may be used for various tracking purposes to decide when to issue commands to the user. In another embodiment, a second loop
5 may be created upon the beginning of a chapter during which information on a next chapter is retrieved.

The "call back" object (*LayerSink*) is thus responsible for creating and communicating with the *LayerImpl* component. The *LayerImpl* component acts as a
10 scheduler, determining when to issue commands to the user.

LayerImpl will issue different DVD commands, based on the type of decoder the user has in their PC. *LayerImpl* will differentiate between the decoders by using the decoder information submitted from the client. The *LayerImpl* will pass the correct
15 DVD command to the client, based on the decoder's capabilities. For example, if the decoder does not support the *TimePlay* event, then the server may send a *ChapterPlay* event and wait appropriately.

The following is an enumerated summary of the steps the component uses to
20 determine when the users will receive the DVD commands:

1. Retrieves the current time, and the time the event starts and ends.
2. Calculates the length of the event.
3. If the event is within a threshold period (i.e. the time users can be queued before
25 the event), then store the first DVD command in memory. Also, store the Chapter information in memory.
4. Create a loop that processes request until the event has completed.
5. In the loop, calculate the lapsed time of the event.
6. In the loop, retrieve the next chapter information.
- 30 7. Create another loop that will loop until time for the next chapter to be played.
8. When the next chapter is ready to play, send the command that was retrieved from the Chapter table.

Business Layer Component

Figure 9 is a flowchart delineating a method for identifying a plurality of events which are played back simultaneously on a plurality of networked client apparatuses. This features is important since a host server may be synchronizing more than one event at once, or during overlapping times. Such events must therefore be distinguished.

10 First, in operation 900, a plurality of events are stored in memory on a plurality of client apparatuses. Each of the events is assigned a unique identifier which is stored in the memory.

In operation 902, the client apparatuses are adapted to be coupled to a host computer via a network, as set forth hereinabove. In operation 904, the identifier of the event which is stored in the memory of the client apparatuses is then retrieved utilizing the network. Such identifier is subsequently compared with an identifier of a scheduled event, as set forth in operation 906. If the comparison renders a match, the playback of the event is begun on the appropriate client apparatuses. Note operation 908.

20 *CbusinessLayer* thus differentiates events by the disk and location ids, uploaded by the client to guarantee backwards compatibility. As set forth earlier, late arrivals can always re-sync with the event.

25 Configuration Manager Component

Figure 10 shows a flowchart delineating a technique for identifying playback devices of a plurality of client apparatuses which are networked to simultaneously playback an event. The present technique is important since the playback devices of the various client apparatuses may differ in make and model. Thus, different commands are required therefor.

In operation 1000, a type of the playback devices of the client apparatuses is first identified. Such "type" may refer to a make, model, or any other distinguishing characteristic of the particular playback devices. A command associated with the identified type of the playback device is then looked up in a look-up table. Note
 5 operation 1002. Such table may be located at the host server, or at any other location such as the client apparatuses.

Thereafter, in operation 1004, the command is sent to the corresponding client apparatus for beginning the playback of the event simultaneously with the playback
 10 of the event on each of the remaining client apparatuses.

This component is thus responsible for identifying what type of reference player is hosting the event. The reference player can be the database, which contains the DVD commands or a real time player. When the initial DVD is command is
 15 requested, the "Synchronizer" table is queried for the host type. From that point forward, the scheduler would know from whom to receive data.

DBConnect Component

20 This component is responsible for communicating with the Synchronizer tables, and for providing access methods for the retrieved data. All interaction from the tables is on a read-only basis. The *LayerImpl* component communicates with this component to retrieve DVD commands and event information.

25 Even though current implementation may be based on a Microsoft platform, hard dependencies on Microsoft or any other 3rd-party development tools may be avoided. To address such issues, the following considerations may be made throughout the code:

30 MFC specific code may be avoided. Instead, STL may be used. ATL and/or MFC code may be encapsulated into separate classes and portioned from the rest of the code. Class implementations may use aggregation pattern to delegate business logic

to the portable classes. Database connection classes may be separated and the communication protocol may be separated with respect to portability to Oracle and other platforms.

5 Figures 11 and 12 illustrate the order of events among the various components of the present invention. In particular, Figure 11 illustrates the manner in which a layer factory is created. As shown, an event is first checked in a database server after which a business layer is created in a WEB server in a manner set forth hereinabove. The foregoing components are then created. Figure 12 illustrates the manner in
10 which user requests are processed. As shown, communication is afforded with the video player on the client machine by means of JAVAscript and JAVA applets. The WEB server, in turn, communicates DVD commands to the video player via the JAVA applets, and also interfaces the database server via the various components thereof which were set forth hereinabove.

15

Alternate Embodiments

To support future enhancements, further components may be included with extendibility as the major objective. Various future enhancements of the product and how they will
20 be addressed will now be set forth.

Hosted Real Time Players

While spirals may retrieve pre-recorded DVD commands from the database,
25 alternate spirals may support a consumer as a host. The architecture may also support plug-in components. Alternate spirals may support the RealTimeConnector component, which accepts host user request and forwards them to the clients. The instant architecture supports the DBConnector which accepts events from the database.

30

Keep Alive Connections

Clients may maintain connections throughout the event. This allows the host to send a various number of commands to the client of the event. Although the spiral disconnects users once a PLAY command has been issued, the Synchronizer class (which will be set forth later) adds each connection to a Thread Pool. This pool of connections can be left open during the life of the event.

Logging Participants

Each request may be logged into the database to provide a reference for the future.

DVD Positioning

As an option, connections may be pooled to allow the synchronization server to direct consumer's machines to the certain locations throughout the entire event.

Synchronization events in alternate spirals may be defined as a combination of play from location event and the actual event. This way, one describes each event in the unambiguous way on the client side and synchronizes it with the server. For example, a situation may be considered where one fast forwards after a movie is played for 15 min and thereafter plays the scene in the movie. In such situation, one has to submit the information to the client player, indicating that it (player) has to start time play from 15 min into the movie and fast-forward to the certain location. A better way would be to analyze what is the next event after fast forwarding occurred and perform a combination for the play from location and next event. This design would require significant changes to the client infrastructure, including *video object*, *remoteagent* and *provider* and should be taken into consideration in any alternate client design.

Classes/Component Diagrams

Figures 13-16 illustrate various class/component diagrams. In particular, Figures 13-16 illustrate a Synchronizer Class Diagram 1300, LayerImpl Class Diagram 1400,

Business Layer Class Diagram 1500, and DBConnect Class Diagram 1600, respectively.

Sequence Diagrams

5

Figure 17 illustrates a logical sequence diagram 1700. As shown, when the server receives a user request, it analyzes the authentication information of the request (date/time, disc id, user id, and BCA number) and the appropriate synchronization event stored in the database. The database contains an *event start threshold* value measured in milliseconds. This threshold defines the amount of time prior to an event that a consumer is eligible to “connect” for the start of the event.

10

If the date/time of the user request lies within the event start threshold, the user is put into wait queue and receive the appropriate data when the time elapses. Note steps 1,2,3,5,6,7 of the Logical Sequence diagram. Otherwise, a message is sent informing the user when the event will occur. Note step 4 of the Logical Sequence diagram.

15

Server side collaboration diagram

20

Figure 18 illustrates a logical sequence diagram 1800 that shows server side collaboration. As shown, server ISAPI extension receives a BroadcastVideoEvents request. It calls *IA_BusinessServer* via *BeginProcess*, to retrieve configuration information. Configuration information contains a playback connector. Playback connector identifies whether the server will have to communicate with a reference player or will it perform playback from the database.

25

At step 6, ISAPI extension will call *IA_BusinessServer CompareTime* method and based on the results will send to the user a predefined web page indicating to retry later or return control to the web server, notifying it (web server) to keep the connection open. At this point connection is pooled and will be processed by the *IA_BusinessServer* at a time of the event.

30

Client Collaboration Diagram

Figure 19 illustrates a logical sequence diagram 1900 showing client side
5 collaboration in accordance with one embodiment of the present invention.

Classes/Interfaces Definition

Definitions of one embodiment of the various classes associated with the software which
10 implements the present invention will now be set forth.

Class Applet1

Purpose:

15

This is the class that implements the applet. The browser will use it to bootstrap our applet.

Responsibilities:

20

- Request a BroadcastVideo event and to gather event status information.

Collaborations:

25 BroadcastEvent, CITIEncrypt

Base class and implemented interfaces:

Javax.Applet

30

Public interface:

Return type: String

Parameters: void

Pre-conditions: None.

Post-conditions: None.

5

GetStartTimeDay Returns the time the event starts in days.

Return type: String

Parameters: void

Pre-conditions: None.

10 Post-conditions: None.

GetStartTimeMnth Returns the time the event starts in months.

Return type: String

Parameters: void

15 Pre-conditions: None.

Post-conditions: None.

GetStartTimeYr Returns the time the event starts in year.

Return type: String

20 Parameters: void

Pre-conditions: None.

Post-conditions: None.

GetLenOfEvent Returns the length of the event.

25 Return type: String

Parameters: void

Pre-conditions: None.

Post-conditions: None.

30 **GetExpiredTime:** Returns lapse time of the event.

Return type: String

Parameters: void

000210" T0968450

Pre-conditions: None.

Post-conditions: None.

getServerTime: Returns the servers current time <SS:MM:HH:DD:MM:YYYY>.

5 Return type: String

Parameters: void

Pre-conditions: None.

Post-conditions: None.

10 **getServerTimeSec:** Returns the servers current in seconds.

Return type: String

Parameters: void

Pre-conditions: None.

Post-conditions: None.

15

getServerTimeMin: Returns the servers current in minutes.

Return type: String

Parameters: void

Pre-conditions: None.

20 Post-conditions: None.

getServerTimeHr: Returns the servers current in hours.

Return type: String

Parameters: void

25 Pre-conditions: None.

Post-conditions: None.

getServerTimeDay: Returns the servers current in day.

Return type: String

30 Parameters: void

Pre-conditions: None.

Post-conditions: None.

000270" T0963460

getServerTimeMnth: Returns the servers current in month.

Return type: String

Parameters: void

5 Pre-conditions: None.

Post-conditions: None.

getServerTimeYr: Returns the servers current in year.

Return type: String

10 Parameters: void

Pre-conditions: None.

Post-conditions: None.

startProc: Calls the ISAPIs "ServerInfo" method.

15 Return type: void

Parameters: String disk id, String location id

Pre-conditions: None.

Post-conditions: None.

20 **msgEvent:** Calls BroadCastEvent applet.

Return type: void

Parameters: void

Pre-conditions: None.

Post-conditions: None.

25

Class BroadCastEvent

Purpose:

30 This is the class that invokes the Synchronizer.

Responsibilities:

- Sets the JavaScript with the command returned from the server.

Collaborations:

5

CITIEncrypt

Base class and implemented interfaces:

10 Java.Thread

Class CDBConnect

Purpose:

15

This is the class provides a public interface for components to request information from the DB tables.

Responsibilities:

20

- Opens the database and Synchronizer, Chapter_Disk tables.
- Queries the Synchronizer by the specified disk id and location id.
- Queries the Chapter_Disk by disk id.
- Provides the next chapter that is scheduled to play.
- 25 ▪ Queries the Decoder_Capabilities table to determine if the requested player is time or chapter play.

Collaborations:

30 DBSyncSet

DBReferenceSet

CDBChapterSet

000210" F0968+150

1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2040 2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105 2106 2107 2108 2109 2110 2111 2112 2113 2114 2115 2116 2117 2118 2119 2120 2121 2122 2123 2124 2125 2126 2127 2128 2129 2130 2131 2132 2133 2134 2135 2136 2137 2138 2139 2140 2141 2142 2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153 2154 2155 2156 2157 2158 2159 2160 2161 2162 2163 2164 2165 2166 2167 2168 2169 2170 2171 2172 2173 2174 2175 2176 2177 2178 2179 2180 2181 2182 2183 2184 2185 2186 2187 2188 2189 2190 2191 2192 2193 2194 2195 2196 2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213 2214 2215 2216 2217 2218 2219 2220 2221 2222 2223 2224 2225 2226 2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2250 2251 2252 2253 2254 2255 2256 2257 2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294 2295 2296 2297 2298 2299 2300 2301 2302 2303 2304 2305 2306 2307 2308 2309 2310 2311 2312 2313 2314 2315 2316 2317 2318 2319 2320 2321 2322 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332 2333 2334 2335 2336 2337 2338 2339 2340 2341 2342 2343 2344 2345 2346 2347 2348 2349 2350 2351 2352 2353 2354 2355 2356 2357 2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429 2430 2431 2432 2433 2434 2435 2436 2437 2438 2439 2440 2441 2442 2443 2444 2445 2446 2447 2448 2449 2450 2451 2452 2453 2454 2455 2456 2457 2458 2459 2460 2461 2462 2463 2464 2465 2466 2467 2468 2469 2470 2471 2472 2473 2474 2475 2476 2477 2478 2479 2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517 2518 2519 2520 2521 2522 2523 2524 2525 2526 2527 2528 2529 2530 2531 2532 2533 2534 2535 2536 2537 2538 2539 2540 2541 2542 2543 2544 2545 2546 2547 2548 2549 2550 2551 2552 2553 2554 2555 2556 2557 2558 2559 2560 2561 2562 2563 2564 2565 2566 2567 2568 2569 2570 2571 2572 2573 2574 2575 2576 2577 2578 2579 2580 2581 2582 2583 2584 2585 2586 2587 2588 2589 2590 2591 2592 2593 2594 2595 2596 2597 2598 2599 2600 2601 2602 2603 2604 2605 2606 2607 2608 2609 2610 2611 2612 2613 2614 2615 2616 2617 2618 2619 2620 2621 2622 2623 2624 2625 2626 2627 2628 2629 2630 2631 2632 2633 2634 2635 2636 2637 2638 2639 2640 2641 2642 2643 2644 2645 2646 2647 2648 2649 2650 2651 2652 2653 2654 2655 2656 2657 2658 2659 2660 2661 2662 2663 2664 2665 2666 2667 2668 2669 2670 2671 2672 2673 2674 2675 2676 2677 2678 2679 2680 2681 2682 2683 2684 2685 2686 2687 2688 2689 2690 2691 2692 2693 2694 2695 2696 2697 2698 2699 2700 2701 2702 2703 2704 2705 2706 2707 2708 2709 2710 2711 2712 2713 2714 2715 2716 2717 2718 2719 2720 2721 2722 2723 2724 2725 2726 2727 2728 2729 2730 2731 2732 2733 2734 2735 2736 2737 2738 2739 2740 2741 2742 2743 2744 2745 2746 2747 2748 2749 2750 2751 2752 2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2768 2769 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781 2782 2783 2784 2785 2786 2787 2788 2789 2790 2791 2792 2793 2794 2795 2796 2797 2798 2799 2800 2801 2802 2803 2804 2805 2806 2807 2808 2809 2810 2811 2812

5 *Public interface:*

Post-conditions: **None.**

Post-conditions: None.

Post-conditions: **None.**

Post-conditions: None.

Return type: String

Parameters: long **, long **

Pre-conditions: None.

Post-conditions: None.

5 Class CConfigMgrImpl

Purpose:

This is the class provides a public interface for components to determine the type of
10 reference player hosting the event.

Responsibilities:

- Opens the database and Synchronizer, Chapter_Disk tables.
- 15 ▪ Queries the Synchronizer by the specified disk id and location id.
- Stores the reference player type.

Collaborations:

20 CConfigMgrRecSet

Base class and implemented interfaces:

Public interface:

25

get_hostType: Returns the reference player host type.

Return type: String

Parameters: short

Pre-conditions: None.

30 Post-conditions: None.

Class threadFunctor

000210" T0968460

Purpose:

This class provides a threading model that classes can use to derive.

5

Responsibilities:

- Calls the CreateEvent function, which opens a named or unnamed event objec.
- 10 ▪ Calls _beginthread, which creates a thread begins execution of a routine at start_address. The routine at start_address must use the __cdecl calling convention and should have no return value. When the thread returns from that routine, it is terminated automatically.
- 15 ▪ Calls the WaitForSingleObject function, which checks the current state of the specified object. If the object's state is nonsignaled, the calling thread enters an efficient wait state.
- Calls the ResetEvent function, which sets the state of the specified event object to nonsignaled.
- 20 ▪ The state of an event object remains nonsignaled until it is explicitly set to signaled by the SetEvent or PulseEvent function.

Collaborations:

CConfigMgrRecSet

25

Base class and implemented interfaces:

Public interface:

30 **start:** Starts the thread.

Return type: void

Parameters: void

addrequest: Adds the request to its vector.

Return type: void

Parameters: void

5 Pre-conditions: None.

Post-conditions: None.

getBLayerInfo: Responsible for getting information about the event.

Return type: void

10 Parameters: std::string&,std::string&, ChttpServerContext*

Pre-conditions: None.

Post-conditions: None.

Class factorySink

15

Purpose:

Manages the layerSink and businessLayerProp objects.

20 *Responsibilities:*

- Stores a layerSink object.
- Returns the "businessssLayerProp" <Business Layer Properties>
- Creates the "businessLayerProp" <Business Layer structure>

25

Collaborations:

LayerSink

businessLayerProp

30

Base class and implemented interfaces:

Public interface:

construct: Stores a layerSink object..

Return type: void

5 Parameters: void

Pre-conditions: None.

Post-conditions: None.

notifyCreateLayer: Responsible for creating a "businessLayerProp".

10 Return type: void

Parameters: BSTR, BSTR, DATE, DATE, LONG

Pre-conditions: None.

Post-conditions: None.

15 Class layerSink

Purpose:

layerSink represents a sink interface and stores a queue of requests. It creates a
20 connection point object.
This call back object, allows asynchronously processing.

Responsibilities:

- 25
 - Acts as the client sink object.
 - Sends the results to the user
 - Creates the "BusinessLayer" and makes it a connection point object.
 - Closes the users connection.
 - Creates a Factory interface by calling "createFactory".
- 30
 - Creates a connection point for the factory.
 - Stores the LayerSink in the FactorySink object.

- Creates a connection point (call back) by calling AtlAdvise, between the connection point container and the client sink object. This allows the client to receive events.
- Calls the connectable objects "getServerLayer". This method fires an event to the clients sink object.
- Create a business layer,
- Store the request in its vector.
- Release the Sink Object (client)
- Calls AtlUnadvise to terminates the ability of the client to receive events.

Collaborations:

Base class and implemented interfaces:

Public interface:

construct: Creates a connection point.

Return type: void

Parameters: void

Pre-conditions: None.

Post-conditions: None.

addRequest: Adds the request to its vector.

Return type: void

Parameters: BSTR, BSTR, DATE, DATE, LONG

Pre-conditions: None.

Post-conditions: None.

createBusinessLayer: Creates a business layer. Create the connection point .

Return type: void

Parameters: businessLayerProp &

Pre-conditions: None.

Post-conditions: None.

updatetime: This call back function translates the time and sends the command to the user.

5 Return type: void

Parameters: long,long

Pre-conditions: None.

Post-conditions: None.

10 Class CBusinessLayer

Purpose:

Creates a layerthread object. This object is responsible for providing access methods,
15 which provide event information.

Responsibilities:

- 20 ▪ The "Synchronizers" createBusinessLayer method creates a class object from the "IBusinessLayer" interface. <The class object is part of the LayerImpl project>
- The BusinesLayers class object <m_ilyer> calls its "Initialize" method. <Note: m_ilyer is the connection point object. It identifies the "Sink Interface".
- 25 ▪ It then calls the "Initialize" method of the connection point.
- The "Initialize" method then calls the "ChkValidEvent" method, which then creates a layerthread object.

Collaborations:

30

CBusinessLayer

layerthread

0002FD" T0968460

Base class and implemented interfaces:

Public interface:

5

Initialize: Calls the "ChkValidEvent" method which kicks of a layer thread.

Return type: void

Parameters: void

Pre-conditions: None.

10 Post-conditions: None.

Class layerthread

Purpose:

15

This object acts as a scheduler, processing request from its queue.

Responsibilities:

20

- Send DVD commands to the user.
- "Syncs" up late comers to the events.

Collaborations:

25

CBusinessLayer

CDBConnect

Base class and implemented interfaces:

30

Public interface:

startThread: Processes requests from the queue

000270" T0968+50

Return type: void

Parameters: void

Pre-conditions: None.

Post-conditions: None.

5

Class CLayerFactory

Purpose:

- 10 This object manages businesslayer objects. Business layer objects communicate with the reference player and notify the user which DVD command to play.

Responsibilities:

- 15
- Send DVD commands to the user.
 - "Syncs" up late comers to the events.
 - This object Implements the IID_LayerFactory interface.
 - This COM object is the servers Connectable Point object.
 - This server object supports connections to sink interfaces. These sink
- 20 interfaces reside on the client side and are equivalent to the "call back" functions in Windows.

Collaborations:

- 25 CBusinessLayer
CDBConnect

Base class and implemented interfaces:

- 30 *Public interface:*

000210" T0958460

